Pachi: State of the Art Computer Go Program

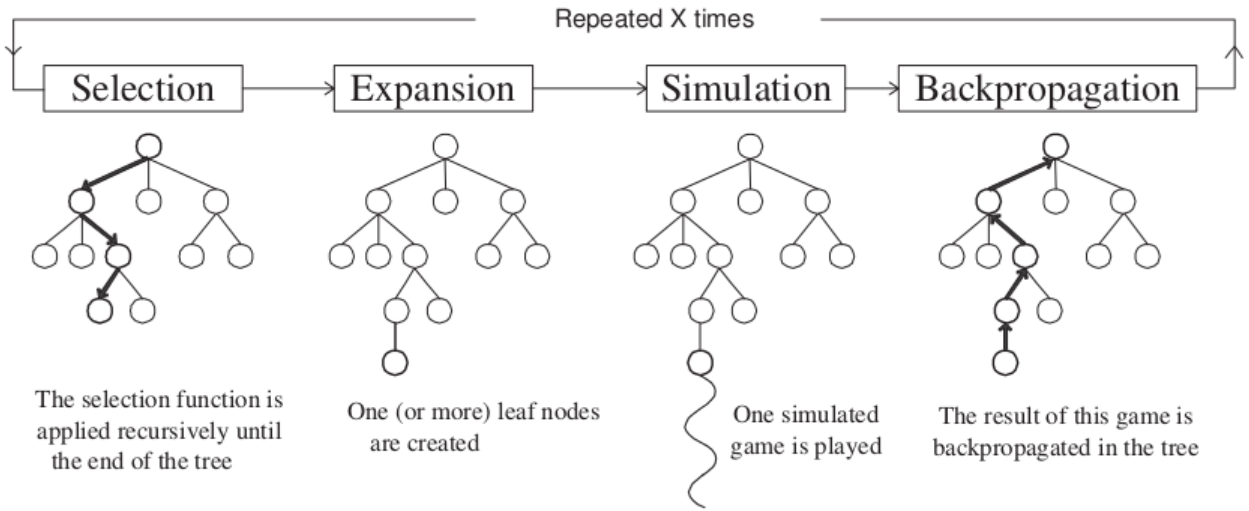
Petr Baudiš, Jean-loup Gailly

ACG13

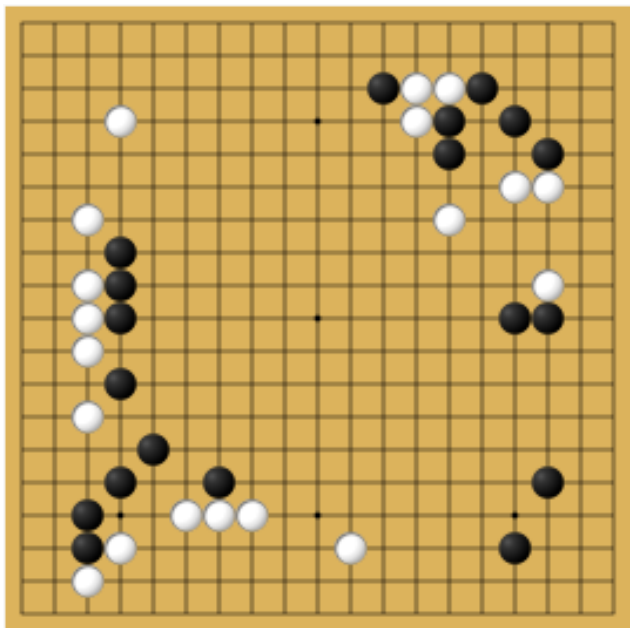
November 2011

Outline

Computer Go and MCTS



(Chaslot, et al.: Parallel Monte Carlo Tree Search)



Our Work, This Presentation

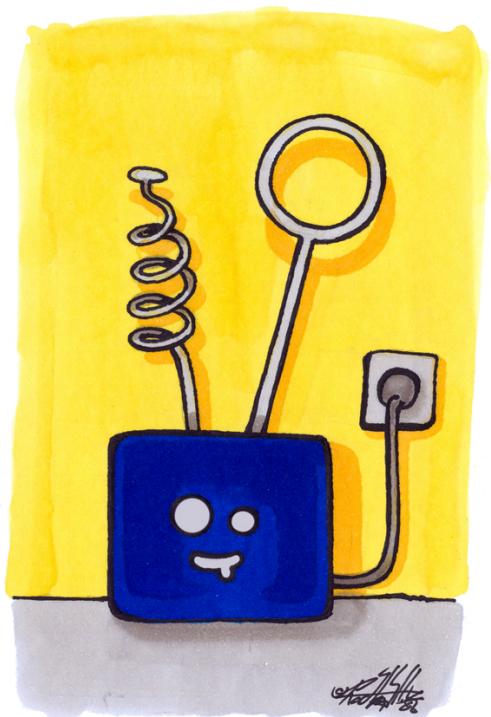
- A nice Go-playing software
- "Engineering-focused Review" — particular high-performance mix of published results
- New improvements — time control, criticality, dynamic komi
- Parallelization and scalability
- A plead to fellow researchers

Outline

Pachi — The Software

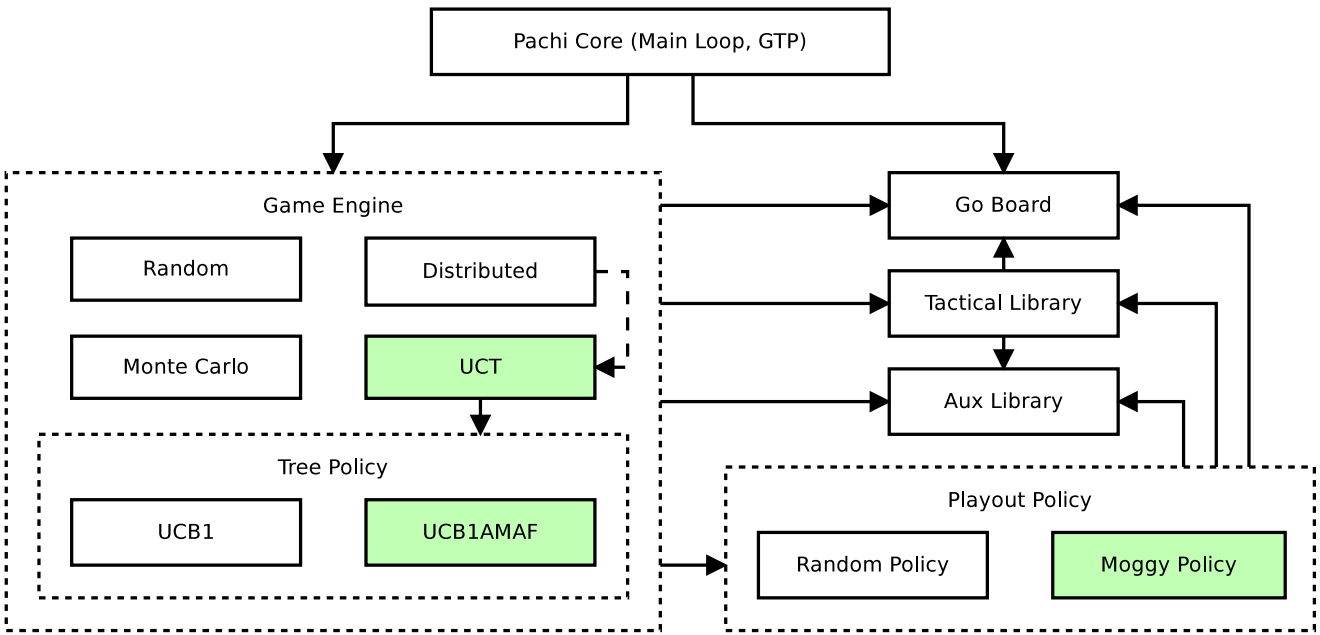
- Artificial intelligence for the game of Go
- Fully open source (GPLv2), including the infrastructure
- Regularly plays on the internet; KGS: 3 dan on cluster, 1 dan on single machine
- 1 thread weaker than Fuego, but scales better (multi-threaded $\geq$ Fuego)

- Support for distributed computation
- Support for some game analysis features
- Not that user friendly



Pachi — The Software

- About 17000 lines of richly comented C code
- Modular, but not with too many abstraction layers
- Easy to add new features, priors, heuristics
- Well-tuned (maybe a bit overtuned), highly configurable



Outline

Feature Mix

- Most of the popular Go heuristics and techniques

- RAVE, no UCB term, prior values for nodes
- Rule-based playouts (like Mogo), 3×3 patterns
- No progressive bias or unpruning
- No probability distribution or large patterns in playouts

Feature Performance

Table: Elo performance of various prior value heuristics on 19×19.

Heuristic	Low-end	Mid-end	High-end
w/o CFG distance prior	-66 ± 32	-66 ± 16	-121 ± 16
w/o playout policy prior	-234 ± 42	-196 ± 16	-228 ± 16
w/o Capture/escape rule	-563 ± 106	-700	-949
w/o 3×3 pattern rule	-324 ± 37	-447 ± 34	-502 ± 36

Lessons learned: The opponent, available time and board size matter

Outline

Time Management

- Naive: Divide time to time-slots based on expected game length
- Spend most time in the middle game (see Erica)

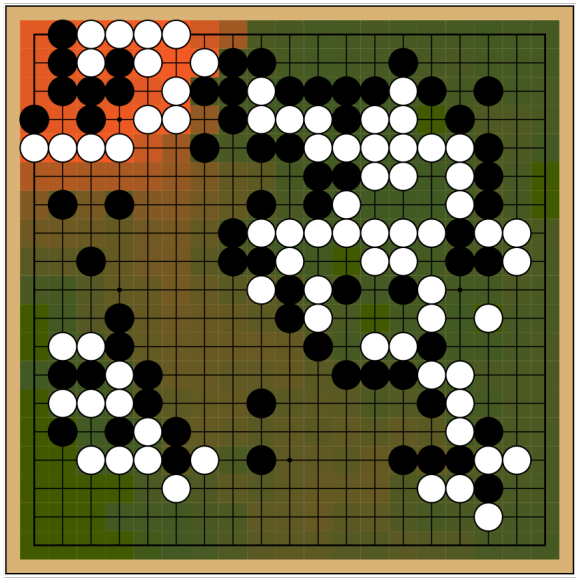
- Spend little on clear moves
- Spend long on moves with unclear followup (multiple good candidates)
- Spend long on moves with unsettled followup (good candidate has bad followups)

- Exact mechanism can be improved
- Low hanging fruit: 80 Elo improvement, and room for more!

Dynamic Komi

- Increase evaluation precision in situations with extreme μ
- **Handicap games:** Linear dynamic komi
  Set komi to $\frac{n}{200} \cdot 8 \cdot h$
  Great for MCTS as black
- **General:** Situational dynamic komi
  Set komi to maintain $\mu \in [0.4, 0.5]$, use ratchet
  Seemed promising, but scaling problems
- **Current:** "Linear adaptive dynamic komi"
  Linear komi, but try to maintain $\mu \in [0.8, 0.85]$
  Great in handicap, point-hungry in endgame

Criticality



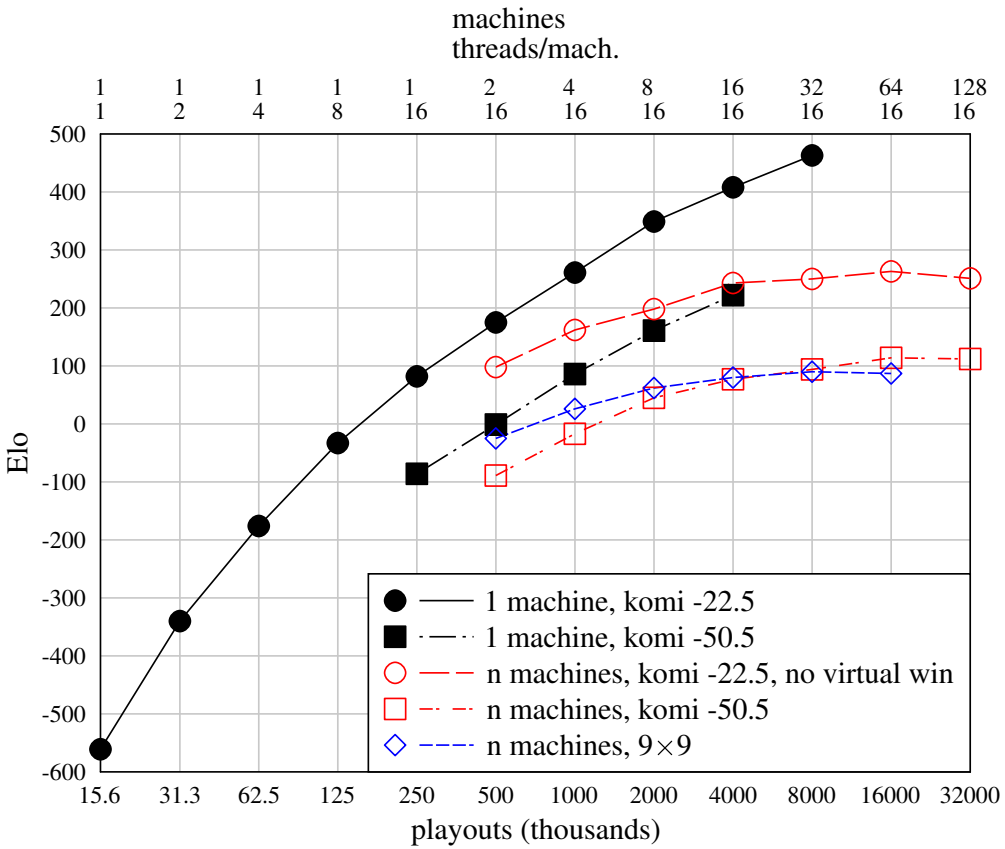
- Covariance of owning a point and winning the game
- Caveats: Owning point $\neq$ playing there; how to integrate in MCTS?
- Previous results with progressive unpruning and plain UCT
- **Our approach:** Add $(C \cdot 1.1 \cdot n_{RAVE})$ RAVE wins
- Seems promising, but scaling problems

(Coulom, R.: Criticality: a Monte-Carlo heuristic for Go programs

Outline

Scalability

- Single machine scales without sign of plateau
- Multiple machines are limited, at most +200 Elo
- Against Fuego 1.1, 500kP/move:



Parallelization

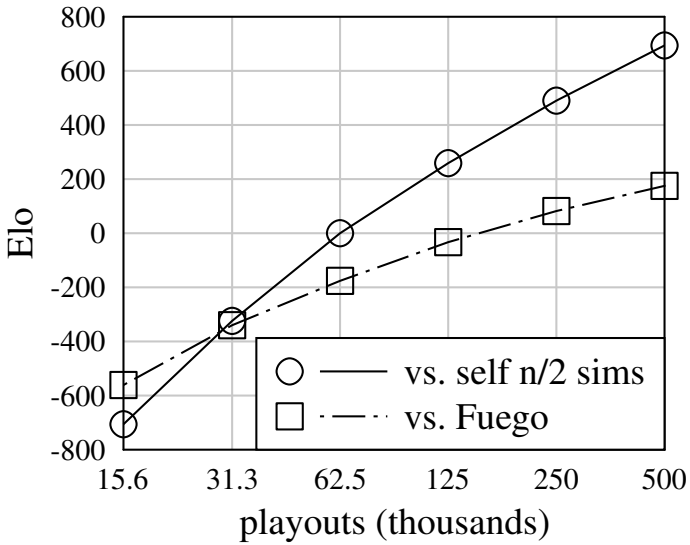
- Single machine: Lockless in-tree parallelization
- Distributed: Root-level synchronization
  (network is just 1 Gb/s Ethernet)

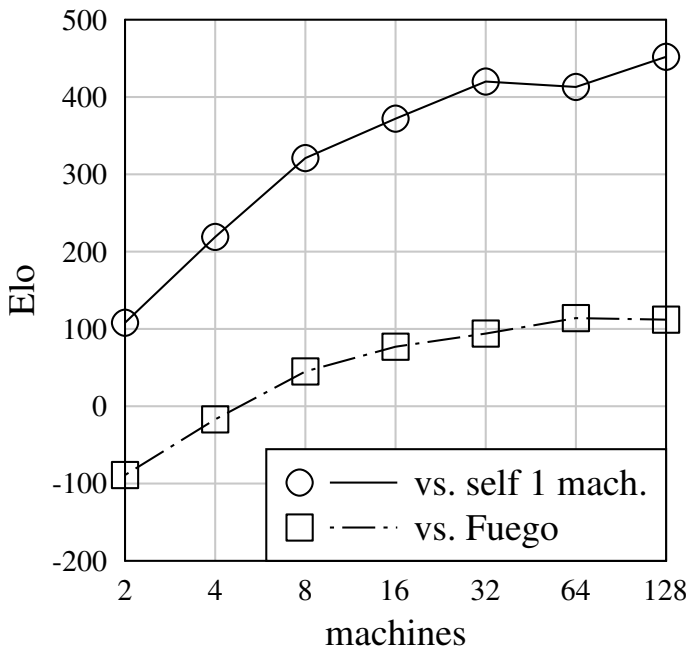
- Virtual loss — 8 losses during descent for thread diversity
- Virtual win — 30 or 5 wins for different tree nodes
  on each machine

Outline

Pleads to the Researchers

- Please avoid self-play experiments
- Please use Elo instead of win percentages
- Please investigate effect on scaling





Conclusion

- Strong program, easy to use for experiments
- Few interesting enhancements inviting further work
- Simple but effective parallelization, good scaling
- It would be nice to improve research reporting